A Survey of Parallel Computing Architectures, Paradigms, and Technologies

The purpose of this document is to compare available parallel computing techniques and their impact on the performance. The document tries to explore the prospects that parallel computing systems promise for increased speed of execution.

1 Parallel Computing Architectures

Parallel computers can be classified into two main categories ([7]):

- Single Instruction Multiple Data (SIMD): all the processing elements receive the same instruction broadcast from the control unit, but operate on different data sets from distinct data streams.
- MIMD (multiple instruction / multiple data): In MIMD machines, the processors are connected to the memory modules by a network.

SIMD architecture is perfectly suited for data parallel problems, as the data can be split into many different independent pieces. Some examples of SIMD machines include Cray Y/MP and IBM9000. The MIMD architecture is capable of running in true "multi-instruction" mode, with every processor doing something different. Some examples of MIMD machines are IBM-SP series, clusters of workstations/PCs, Intel Paragon etc. The MIMD architecture is classified into shared memory and distributed memory architectures based on the memory structure of the system. In shared memory architecture, the same memory is accessible to multiple processors and synchronization is achieved by controlling task's reading from and writing to the shared memory. The advantage of this type of architecture is the ease of programming. Since multiple processors have access to the same memory, however, the scalability of the system is limited by the number of access pathways to memory. To improve the performance, each processor can be provided with a local memory and the global memory is shared among all the processors in the system. Depending on the speed of memory accessing, shared memory computers can be classified into the Uniform Memory Access (UMA) or Non-Uniform Memory Access (NUMA) computers. Some examples of shared memory computers include IBM ES-9000 and SGI Power Challenge.

In a distributed memory parallel computer, each processor element has its own local memory and synchronization is achieved by the communication between processors via an interconnection network. The major concern in a distributed memory architecture is data decomposition, i.e., how to distribute data among processors to minimize processor communication. Distributed memory machines use message passing or distributed shared memory models. In message passing systems, program tasks communicate by explicitly sending/receiving data packages to/from each other. The process of sending/receiving data is specified and controlled by the application program via message passing libraries (PVM, MPI) or languages (VPP-Fortran, Linda). The virtual shared memory systems provide programmer with a shared memory programming model but the actual design of the hardware and the network is distributed. A sophisticated memory mapping scheme ensures that the entire distributed memory space can be uniquely represented as a single shared resource. The actual communication model is at the lowest level and it uses message passing.

The other class of MIMD computers that are based on both shared and distributed memory architectures is the Symmetric Multi-Processor (SMP) clusters. A typical SMP system in fact is a cluster of a large group of nodes as a distributed memory computer. Each node in turn has a small number of processors with shared memory.

Both the shared and distributed memory computers are constructed by connecting processors and memory units using a variety of interconnection networks. In shared-memory systems, interconnection network is for the processors to communicate with the global memory. In distributed system, interconnection network is for processors to communicate with each other. Static interconnection networks consist of point-to-point communication links among processors that are typically used in message-passing computers. In dynamic networks, communication links are connected to one another dynamically by switching elements. Dynamic networks are typically used with shared memory computers. Examples of static networks include completely connected, star, and ring and dynamic network examples include crossbar, bus-based and multistage interconnection networks.

1.1 Shared Memory vs Distributed Memory

There have been a number of studies comparing the performance of parallel programs on shared memory and distributed memory systems. Games and Lee [9] examined the tradeoff between parallel-software development effort and the resulting processing performance for both type of systems for applications in signal and digital image processing. They concluded that both programming approaches showed comparable tradeoffs between the ease of development and the level of performance that can be achieved.

2 Parallel Programming Paradigms

There are a number of parallel programming paradigms:

- Functional Languages (dataflow): Functional languages express computation in terms of pure functions and are amenable to parallel evaluation (Sisal). They require extra overhead in work and storage.
- Data Parallel: Parallelism is not expressed as a set of processes whose interactions are managed by the user, but rather as parallel operations on aggregate data structures (CMF, HPF, HPC++)
- Parallelizing compilers: Usually loop based often with directives
- Shared Memory : multiple threads executing common pool of tasks (pthreads, OpenMP)
- Message Passing: processes communicate using send/receive (PVM, MPI)
- Remote Memory Access: Provides one-sided communications, (put/get), which store and fetch data to or from memory of another processor. Like other shared memory constructs, remote memory access must be used carefully to avoid corruption of shared data.

2.1 Message Passing Standards

There have been a number of efforts to develop and standardize an interface for parallel computing. Foremost in these efforts have been the HPF, PVM [22], and MPI [8] activities. MPI is a portable message passing standard that runs both on tightly coupled, massively parallel processors and on networks of workstations. The motivation for developing MPI was to standardize the message passing API rather than having different proprietary APIs.

PVM is a byproduct of a research project at Oak Ridge National Laboratory and the University of Tennessee. The overall objective of PVM is to enable a heterogeneous collection of computers to be used cooperatively for concurrent or parallel application. The PVM system is composed of two parts. The first part is the system part which is handled through a daemon that resides on all computers making up the virtual machine. The second part of the system is a library of PVM interface routines. This library contains user-callable routines for message passing, spawning processes, coordinating tasks, and modifying the virtual machine. PVM supports a mixture of functional and data parallelism.

Kitowski et al [11] compared PVM and MPI performance for a short-range molecular dynamics simulation problem and they found MPI to be more robust and efficient in comparison with PVM. Their studies were conducted on HP Exemplar SPP1200 and SPP1600 computers, and Intel Paragon multicomputer. Markus et al [18] presented a comparison of several MPI implementations on different hardware platforms based on the performance of PDE solvers from the parallel ELLPACK PSE. The performance of different MPI implementations were comparable and better than that of PVM and other message passing libraries.

2.2 Shared Memory Standards

The early efforts for specifying parallelism was through the creation of concurrent processes using UNIX fork-join constructs. This approach paved way to using lightweight POSIX threads (pthreads), which are much more efficient. OpenMP [5] uses the fork-join model of parallel execution with threads sharing variables. OpenMP simply provides the API to manage the lightweight threads.

Linda [4], introduced in the mid 1980's, implements virtual shared memory (VSM) for supercomputers and workstation clusters. Linda provides an abstraction of a tuple space, a distributed shared memory (DSM), that can be used by processes to communicate and synchronize despite the lack of physical shared memory. Cray provides a shared memory access library called SHMEM that uses the remote memory access paradigm. The protocol is available only on Cray MPP systems.

2.3 Beowulf Clusters and Shared Memory

The linux kernel provides a VFS-like interface to the virtual memory system and researches have tried to exploit this feature to provide page-based network virtual memory or distributed shared memory. Page-based DSMs use the virtual memory hardware of the processors and a software enforced ownership and consistency policy to give the illusion of a memory region shared among processors.

There have been many attempts to emulate shared memory in software on a distributed system, but so far none have resulted in good performance. A group of researchers at Rice University developed a DSM called TreadMarks [1]. It provides a set of facilities and code annotation tools that enable the abstraction of a shared memory system. However, it provides support only for C programming language. Lu et al [13] developed a new compiler that targets a software DSM that can use OpenMP on NOW. They also compared the performance of TreadMarks, OpenMP, and MPI on NOW for a number of different applications. MPI outperformed the shared memory protocols in all cases.

2.4 Performance Comparison - message passing vs shared memory

There have been a number of studies on comparing the performances of message passing and shared memory systems. Luecke and Lin [14] compared the performance of MPI and OpenMP programs on SGI Origin 2000. The MPI implementations performed significantly better than the corresponding OpenMP implementations. They used up to 128 processors in their experiments. Luo [17] presented the comparison of performance of MPI and shared memory on three different shared memory platforms: the DEC AlphaServer 8400/300, the SGI Power Challenge, and the HP-Convex Exemplar SPP1600. The MPI implementation on the SGI Power Challenge and HP System was superior to others, with the shared memory schemes performing slightly better than the MPI schemes on the DEC system. They used vendor specified shared memory schemes in their studies.

Some studies have tried to exploit the shared memory architecture present in the SMP clusters by using a hybrid MPI-OpenMP approach [20, 21]. Despite the perceived reduction in communication time, the hybrid codes did not perform as well as the equivalent message passing code or shared memory code and in some cases, the performance actually deteriorated. Chow and Hysom [6] concluded that the use of hybrid techniques is not trivial, is highly application dependent, and requires careful analysis of cache and memory utilization.

Luecke et al [15] studied the performance and scalability of SHMEM and MPI-2's remote memory access routines for different communication patterns on SGI Origin 2000 and a Cray T3E. They reported that the SHMEM routines outperformed MPI-2's routines for all the tests conducted and SHMEM routines exhibited better scalability.

2.5 MLP - Multi Level Parallelism

Taft [23] presented a new computing technique for parallel computing called multi-level parallelism. The development of this technique was motivated by new hardware designs that are closer to true shared memory architectures. The inherent limitation with shared memory programming is the overhead of memory latency and synchronizations. MLP tries to overcome these limitations by identifying multiple levels of parallelism in the code: coarse grained, task parallelism and fine grained, data parallelism. All communication is through true shared memory references between independent processes. They applied this technique to the CFD code OVERFLOW and reported linear, sustained performance up to 512 processors with almost 3 fold improvement in MPI's performance. The technique has applicability to many CFD and other vector codes. The use of this technique currently is limited to shared memory architectures and the extension to cluster environments is under development.

2.6 Scalability

In the study conducted by Luecke and Lin [14], the scalability of MPI and OpenMP programs were compared on SGI Origin 2000. They reported that MPI implementations scaled consistently better than the corresponding OpenMP implementations. Luecke et al [16] also studied the scalability of MPI on different systems, including an NT Cluster, a Myrinet Linux cluster, an Ethernet Linux cluster, a Cray T3E-600, and a SGI Origin 2000. The scalability and performance of MPI programs was found to be best on the SGI Origin 2000. Among the different clusters compared, the Ethernet Linux cluster outperformed the other two.

Figure 1 compares the performance of a number of parallel systems for the atmospheric research mesoscale model 5 (MM5) [19]. All the systems used MPI implementations. Timings for the Pittsburgh Supercomputer Center Terascale Computing System (PSC TCS) was done using straight-MPI (MPI over shared memory for communication within nodes and using MPI-over Quadrics for communication between nodes). They were able to achieve 105Gflops/s by using 512 processors. The AlphaServerSC is an SMP system and the model was run on it using straight-MPI (similar to the PSC TCS) and this system provided approximately 40Gflops/s with 512 processors. The Fujitsu VPP5000 is a distributed-memory machine with vector processors. The model was run by using Fujitsu's implementation of MPI. The IBM SP WH2 timings provided approximately 25Gflops/s with 256 processors. The HPTi ACL/667 is an Alpha Linux cluster at NOAA. The model was run using MPI-over Myrinet. The Pentium-III ScaliMPI timings were conducted on 16 dual 800Mhz Pentium-III nodes and using ScaMPI.

3 Parallel Program Design Issues

The design of a parallel algorithm can be viewed as consisting of four stages. The first stage consists of decomposing the problem into evenly sized fine-grained tasks to maximize potential parallelism. The decomposition can be done based on data (domain decomposition) or based on computation (functional decomposition). The next step is to determine the communication pattern among tasks. The tasks may need to be combined into coarser grained tasks, if necessary, to reduce communication requirements. This step is called agglomeration. Finally, each task is assigned to a processor in a manner that attempts to satisfy the competing goals of maximizing processor utilization and minimizing communication costs.

Factors Affecting Performance

Load balancing: is used to distribute computations fairly across processors in order to obtain the highest possible execution speed. Load balancing can be carried out statically or dynamically .

Concurrency: defines the work done simultaneously.

Overhead: represents the work not present in serial computation. Time spent in communication, synchronization, and idling contributes to the overhead.

Coping with memory latency: use of caches, local memory, low-latency network, fast interface other techniques such as prefetching, block transfer etc. are also employed.

3.1 Parallel Performance Modeling

The performance of a parallel program is a complex and multifaceted issue. The performance metrics to measure the performance of a parallel program includes execution time, parallel efficiency, memory requirements, throughput, latency, network throughput, portability, and scalability.

Scalability refers to the effectiveness with which parallel algorithm can utilize additional processors. Algorithm is scalable if its parallel efficiency can be maintained at constant value by increasing problem size, as the number of processors grows.

Communication time is the time spent in sending and receiving messages. Time spent in sending one message can be modeled by

$$T_{msg} = t_s + t_w L \tag{1}$$

where t_s is the startup time for message, t_w is transfer time for word and L is the length of message. The bandwidth of communication channel can be defined as $1/t_w$. The startup cost usually dominates when sending many small messages and bandwidth dominates for large messages.

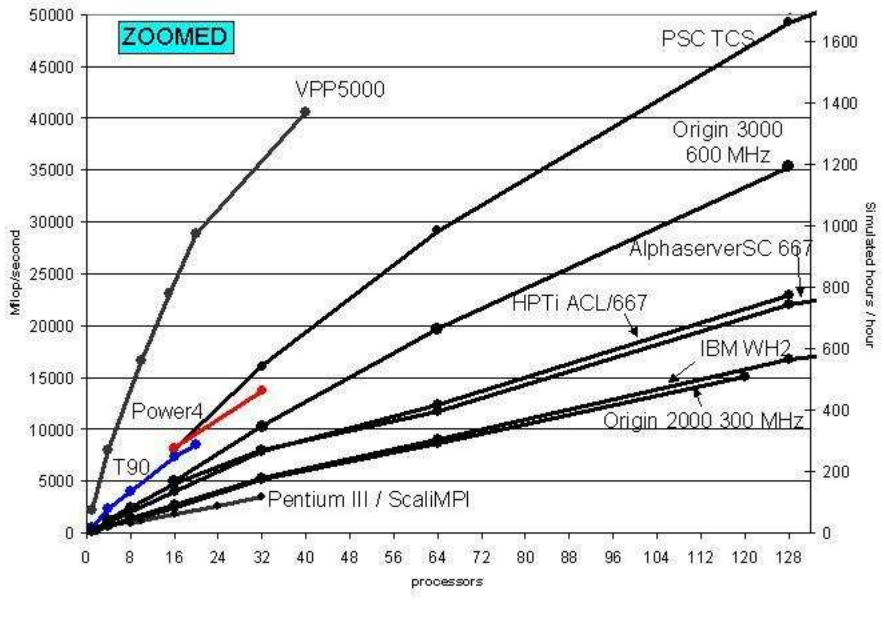

Figure 1: MM5 floating point performance on various platforms (Feb 20, 2002) [19]

3.2 Comparison of Interconnects

Bal et al [2] compared the performance of three different networks, Fast Ethernet, ATM, and Myrinet. The latency on Myrinet was the lowest, followed by Fast Ethernet and ATM. The implementation of a message passing system is based on the interconnect's communication protocol. For example, a Fast Ethernet or Gigabit Ethernet interconnect runs on a TCP/IP-based system. Systems based on Grand Message (GM) and Virtual Interface Architecture (VIA) use Myrinet and Emulex, respectively. The GM and VIA protocols are more efficient than TCP/IP as shown in Figures 2 and 3. Although the figures indicate that the performance of Myrinet and Emulex are better, the study conducted by Leng et al [12] concluded that the Gigabit ethernet provides a better performance to price ratio than the other interconnects. With future reductions in the cost for ethernet interconnects, this ratio is further expected to improve.

3.3 Cluster Design

Figure 4 shows the different design choices for building a beowulf cluster. The comparison of different technologies were discussed in the earlier sections.

4 Conclusions

The following inferences can be made from the review of available parallel computing technologies:

- MPI has emerged as the leading message passing standard
- Emulating shared memory on distributed memory architectures has not so far resulted in good performance
- MPI has emerged as the leading portable parallel computing standard
- Shared memory protocols can be optimized to provide better performance than MPI on shared memory machines, but are highly application specific.
- Hybrid approaches to parallel programming have not been able to beat the performance of pure message passing approaches
- Shared memory programming using MPI-2 is still in infancy
- New techniques such as MLP offer the prospect of achieving a greater performance than that of MPI. The MLP technique is relatively new and its performance on cluster environments needs to be investigated.
- Gigabit Ethernet interconnect provides the best performance/price ratio among the interconnect choices

References

[1] C. Amza, A. L. Cox, S. Dwarkadsa, P. Keleher, H. Lu, R. Rajamony, W. Yu, and W. Zwanenepoel. TreadMarks: Shared memory computing on networks of workstations. *IEEE Computer*, 29(2):18–28, February 1996.

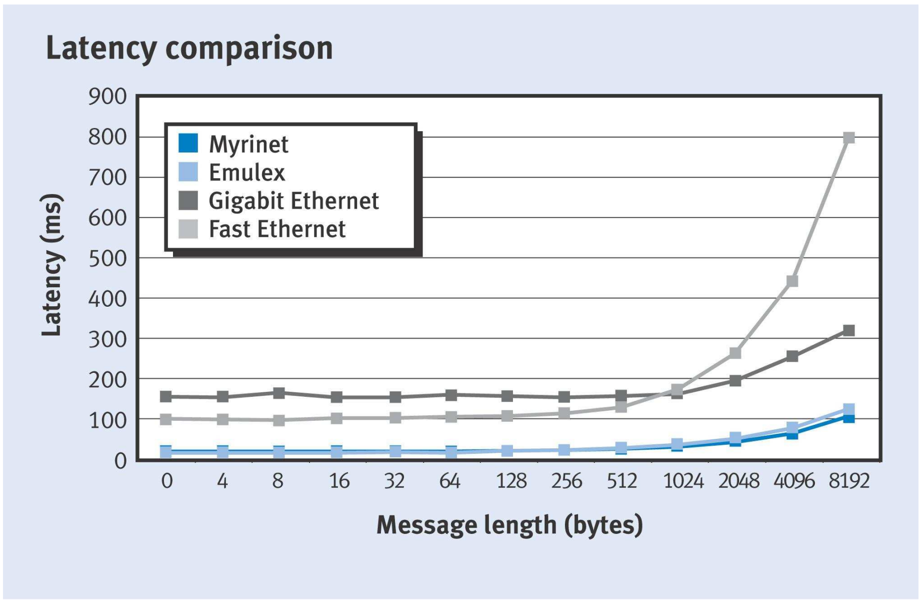

Figure 2: Network Technology Comparison (Latency) [12]

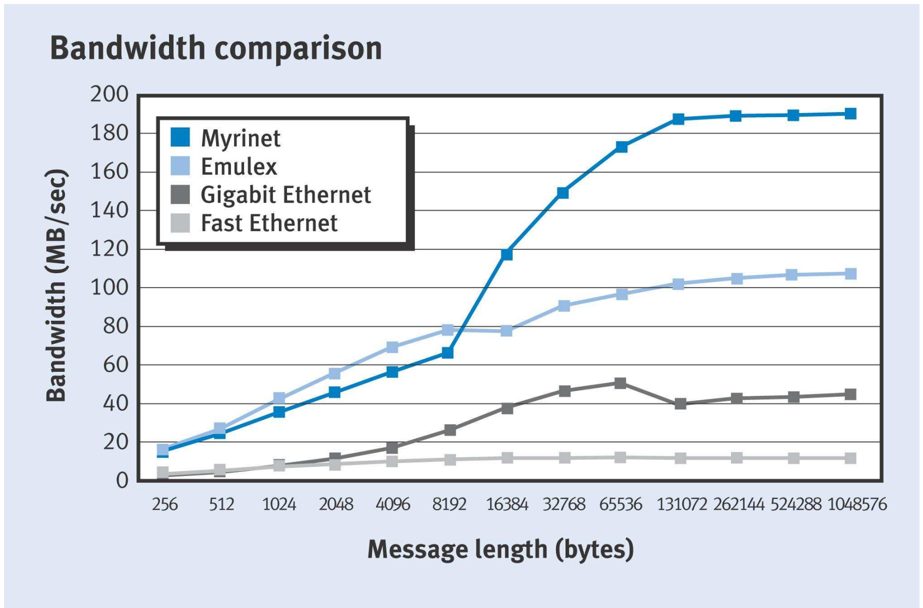

Figure 3: Network Technology Comparison (Bandwidth) [12]

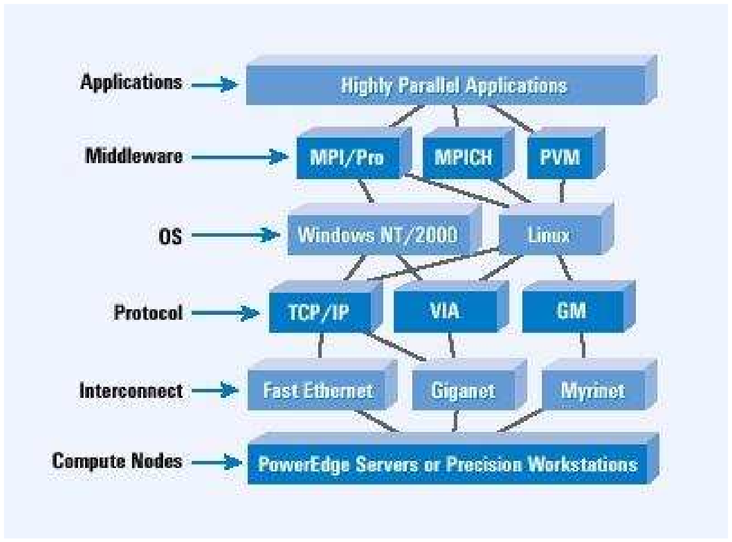

Figure 4: Architectural Stack of a Beowulf Cluster

Table 1: Comparison of MPI vs shared memory programming models [3]

	MPI	Pthreads	HPF	OpenMP
Scalable	Y	N	Y	Y
Incremental Parallelization	N	N	N	Y
Portable	Y	Y	Y	Y
Fortran binding	Y	N	Y	Y
Performance oriented	Y	N	N	Y
Data Parallelism	Y	Y	Y	Y

Table 2: A comparison of the major features of PVM and MPI [10]

PVM	MPI
Virtual machine concept	No such abstraction
Supports heterogeneous NOW and MPP	Intended primarily for MPP, supports NOW.
Simple message passing	Extensive messaging support
Communication topology unspecified	Supports logical communication
PVM implementations interoperable across host architecture boundaries	MPI does not support interoperability
Portability over performance	Performance over portability
Contains resource management, load balancing, process control	Primarily concerned with messaging
Programs in C, C++, or Fortran may freely intercommunicate	Interlanguage communication not supported
Robust fault tolerance	Not supported

[2] H. Bal, R. Hofman, and K. Verstoep. A comparison of three high-speed networks for parallel cluster computing. In D. K. Panda and C. B. Stunkel, editors, *Lecture Notes in Computer Science*, volume 1199, pages 184–197. Springer-Verlag, 1997.

[3] OpenMP Architecture Review Board. OpenMP: A proposed industry standard API for shared memory programming. http://www.openmp.org, October 1997.

[4] N. Carriero and D. Gelernter. *How to Write Parallel Programs.* MIT Press, 1990.

[5] R. Chandra, R. Menon, L. Dagum, D. Kohr, D. Maydan, and J. McDonald. *Parallel Programming in OpenMP.* Morgan Kaufmann, 2000.

[6] E. Chow and D. Hysom. Assessing performance of hybrid MPI/OpenMP programs on SMP clusters. Technical Report UCRL-JC-143957, Lawrence Livermore National Laboratory, Livermore, CA, 2001.

[7] J. J. Dongarra, I. S. Duff, D. C. Sorensen, and H. A. Van Der Vorst. *Numerical Linear Algebra for High Performance Computers.* Society for Industrial and Applied Mathematics, 1998.

[8] MPI Forum. MPI: A message passing interface standard. *International Journal of Supercomputer Application*, 8(3/4):165–416, 1994.

[9] R. A. Games and P. T. Lee. Performance comparison of distributed and shared memory parallel signal and image processing. Technical Report 98B0000085, Center for Integrated Intelligence Systems, Bedford, MA, October 1998.

[10] G. A. Geist and J. A. Kohl. PVM and MPI: A comparison of features. *Calculateurs Paralleles*, 8(2), 1996.

[11] J. Kitowski, K. Boryczko, and J. Moscinski. Comparison of PVM and MPI performance in short-range molecular dynamics simulation. In M. Bubak, J. Dongarra, and J. Wasneiwski, editors, *Proceedings of 4th European PVM/MPI User's Group Meeting, Lecture Notes in Computer Science 1332*, pages 11–16, Cracow, Poland, 1997.

[12] T. Leng, R. Ali, C. Stanton, and J. Hsieh. HPC cluster interconnects and message passing systems: From proprietary to commodity. In *Dell Power Solutions - High Performance Computing*, volume 4, 2001.

[13] H. Lu, C. Hu, and W Zwanenepeol. OpenMP on networks of workstations. In *Proceedings of SC'98*, pages 7–13, Orlando, FL, November 1998.

[14] G. R. Luecke and W. H. Lin. Scalability and performance of OpenMP and MPI on a 128 processor SGI Origin 2000. *Journal of Performance Evaluation and Modeling of Computer Systems*, 2001. in print.

[15] G. R. Luecke, S. Spanoyannis, and M. Kraeva. The performance and scalability of SHMEM and MPI-2 one-sided routines on a SGI Origin 2000 and Cray T3E-600. Iowa State University, October 2001.

[16] G. R. Luecke, J. Yuan, S. Spanoyannis, and M. Kraeva. Performance and scalability of MPI on PC clusters. Iowa State University, January 2000.

[17] Y. Luo. Shared memory vs message passing: the COMOPS benchmark experiment. Technical Report CIC-19, Mail Stop B256, Los Alamos National Laboratory, 1997.

[18] S. Markus, S. B. Kim, K. Pantazapoulos, A. L. Ocken, E. N. Houstis, P. Wu, S. Weerawarana, and D. Maharry. Performance comparison of MPI implementations using the parallel ellpack pse. In *Proceedings of the Second MPI Developer's Conference*, pages 162–169. IEEE Computer Society Press, 1996.

[19] J. G. Michalakes. Penn State/NCAR Mesoscale Model 5 benchmark. http://www.mmm.ucar.edu/mm5/mpp/cowbench/.

[20] D. J. Scales, K. Gharachorloo, and A. Aggrawal. Fine-grain software distributed shared memory on SMP clusters, February 1998.

[21] R. Stets, S. Dwarkadas, N. Hardavellas, G. Hunt, L. Kontothanassis, S. Parthasarathy, and M. Scott. Cashmere-2L: Software coherent shared memory on a clustered remote-write network. In *Proceedings of the 16th ACM Symposium on Operating System Principles*, October 1997.

[22] V. S. Sunderam. PVM: A framework for parallel distributed computing. *Concurrency: Practice and Experience*, 2(4), 1990.

[23] J. R. Taft. Performance of the OVERFLOW-MLP CFD code on the NASA AMES 512 CPU Origin system. Technical Report NAS-00-005, NASA Ames Research Center, March 2000.